EasyChair Preprint

№ 10095

Can Reinforcement Learning Improve Order Decision in Multi-Echelon Inventory Systems? A Linear System Case Study

Mingxuan Sun

May 12, 2023

Can Reinforcement Learning Improve Order Decision in Multi-echelon Inventory Systems? A Linear System Case Study

Indicate Submission Type: Completed Research Paper

Abstract

In this paper, we introduce a novel formulation for the Markov Decision Process (MDP) model specifically tailored for linear inventory systems and present a cutting-edge reinforcement learning (RL) algorithm, termed Shaped-nStep-Double-DQN. By establishing various three-echelon linear inventory systems, we convert the pertinent order placement challenges into optimal policy determination problems within the MDP framework. The experiments demonstrate that the order placement strategies learned by the Shaped-nStep-Double-DQN algorithm in deterministic linear inventory systems are nearly consistent with the optimal order placement strategies, serving as a good approximation. In stochastic linear inventory systems, the ordering strategies learned by the Shaped-nStep-Double-DQN algorithm perform better than the base-stock policy, exhibiting superior inventory performance.

Keywords: Reinforcement Learning, Shaped-nStep-Double-DQN, Linear Inventory Systems, Order Decision

Introduction

Inventory encompasses items held by enterprises for maintenance, production, and resale, spanning raw materials, components, work-in-progress, finished products, equipment, spare parts, and services. An inventory point is a location designated for holding stock, and when multiple inventory points are organized within a system, it becomes a multi-echelon inventory system, as a crucial component of supply chains. Holding inventory provides businesses with several benefits, such as coping with uncertain demand, exploiting economies of scale, and addressing strategic needs. However, it also incurs various holding costs, thus making efficient management of multi-echelon inventory systems crucial. A critical issue in managing these systems is the ordering problem, which determines when and how much to order at each inventory point. The efficacy of order management strategies is paramount for an organization's long-term sustainability and prosperity. And by employing effective ordering approaches, companies can reduce inventory expenses, enhance efficiency, expedite delivery times, bolster operational performance, elevate service quality, augment customer satisfaction, and fortify supply chain resilience.

Over the past several decades, inventory research has centered on addressing ordering challenges in order to achieve superior inventory control[1]. However, despite over 60 years of exploration and accomplishments in resolving a variety of common inventory system issues, optimal strategic solutions for numerous inventory management concerns remain scarce. These constraints predominantly encompass: 1) the lack of solving algorithms for the majority of ordering problems, or the existence of only suboptimal algorithms; 2) challenges in the practical application of algorithms; 3) inadequate computational performance of extant solving algorithms; and 4) elevated academic prerequisites for inventory management personnel. On the other hand, RL has emerged as a prominent field in AI, demonstrating its capabilities and prospects across various industries. From AlphaGo's victory against the world's top Go player, Ke Jie, to DeepMind's AlphaTensor solving a 50-year-old open problem in mathematics, RL has played an indispensable role. Besides, OpenAI's ChatGPT, a chatbot based on the GPT-4 model, also showcases impressive results in professional and academic exams, reflecting an intelligence level close to humans.

In summary, efficient ordering strategies in multi-echelon inventory management constitute a vital aspect of supply chain operations, proving indispensable for the ongoing viability of organizations. RL,

as an AI technology with immense potential, could be a promising solution to tackle existing limitations in ordering and lead to better-performing multi-echelon inventory systems. By integrating RL algorithms into the ordering process, organizations can benefit from more efficient ordering strategies that improve multi-echelon inventory management, thus supply chain resilience, and competitiveness in the global marketplace.

Literature Review

Clark and Scarf's (1960)[1] groundbreaking paper first introduced the concepts of "echelon inventory" and "linear systems," laying the foundation for further research in this area. They transformed high-dimensional optimization problems into a series of nested one-dimensional problems and demonstrated that the optimal ordering strategy is an echelon-base-stock policy. Chen and Zheng (1994)[2] provided an indirect proof, while Muharremoglu and Tsitsiklis (2003)[3] offered a direct proof for the validity of the echelon-base-stock policy. Despite the simplicity of this optimal inventory strategy, calculating the optimal echelon-based stock level can be quite complex due to the intricate cost function. Bertsekas et al. (1997)[4] were the first researchers to explore the application of RL to inventory management. Following their groundbreaking work, Kimbrough et al. (2002)[5] investigated the performance of reinforcement learning-trained agents in the MIT Beer Game and its variants, leading to substantial reductions in supply chain costs and the mitigation of the bullwhip effect. Giannoccaro et al. (2002)[6] delved into a three-echelon linear inventory system characterized by stochastic lead times and demand, formulating a RL algorithm known as "SMART" to tackle this problem. This algorithm outperformed a traditional periodic review base-stock policy in terms of total cost and customer waiting time. Other notable studies in this field include van Tongeren et al. (2007)[7], who employed Q-learning in the Beer Game and demonstrated a significant alleviation of the bullwhip effect, and Chaharsooghi et al. (2008)[8], who applied Q-learning to the same game while using genetic algorithms as a benchmark for performance comparison. Besides, Valluri et al. (2009)[9] investigated the use of linear function estimators, SARSA(λ) methods, and tiling coding TD(λ) methods in a four-level linear inventory system with constant demand. Her results indicated that the TD(λ) method with linear function estimators and tiling coding exhibited superior convergence properties. Mortazavi et al. (2015)[10] evaluated learned ordering strategies through simulation, considering factors such as inventory levels, total inventory costs, and customer waiting times. Kara et al. (2018)[11] compared base-stock policies, Q-learning, and SARSA algorithms in perishable inventory management to understand their respective performances in a single inventory point ordering problem. Boute et al. (2021)[12] offered recommendations for implementing DRL techniques across various inventory systems. Goedhart et al. (2022)[13] apply RL in omni-channel retailing.

Problem Setting

The inventory system investigated in this study is a three-echelon linear inventory system involving a single, non-perishable product. In this system, the inventory points are arranged from downstream to upstream, sequentially consisting of the retailer, warehouse, and manufacturer. External to the system, consumers are associated with the retailer, while suppliers are linked to the manufacturer. Consumers, retailers, warehouses, and manufacturers are only permitted to place orders with their immediate upstream inventory points. We assume that the specific sequence of events in the linear inventory system in one time period:

- Step 1 Shipping: The supplier ships to the manufacturer; the manufacturer ships to the warehouse; the warehouse ships to the retailer; the retailer ships to the consumer.
- Step 2 Receiving goods: The manufacturer receives goods from the supplier; the warehouse receives goods from the manufacturer; the retailer receives goods from the warehouse; the consumer receives goods from the retailer.
- Step 3 Ordering: The consumer orders from the retailer; the retailer orders from the warehouse; the warehouse orders from the manufacturer; the manufacturer orders from the supplier.

- Step 4 Receiving orders: The retailer receives orders from the consumer; the warehouse receives orders from the retailer; the manufacturer receives orders from the warehouse; the supplier receives orders from the manufacturer.
- Step 5 Calculating costs: Calculating the total cost of the inventory system in the current time period.

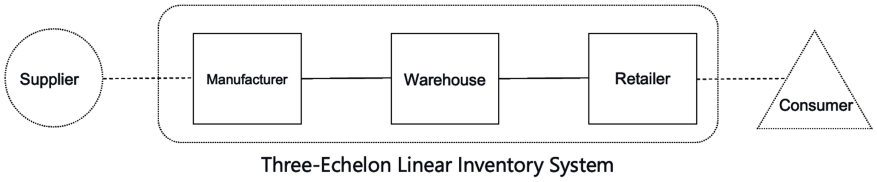

Figure 1: Three-echelon Linear Inventory System

MDP Setting

To effectively implement RL algorithms, a robust Markov Decision Process (MDP) model that captures problem-specific features is crucial. Based on existing literature and linear inventory system features, we develop a novel discrete-time MDP for formalizing the ordering process.

State

Let $\mathrm{IL}_t(i)$ denote the inventory level at inventory point i at time period t when ordering; $\mathbf{IL}_t$ represents the vector of inventory levels at each inventory point at time period t when ordering, i.e., $\mathbf{IL}_t = (\mathrm{IL}_t(1), \mathrm{IL}_t(2), \mathrm{IL}_t(3))$; $\mathrm{IO}_t(i)$ is the amount of goods ordered but not yet received at the inventory point i at time period t when ordering; $\mathbf{IO}_t$ represents the vector of goods ordered but not yet received at each inventory point at time period t when ordering, i.e., $\mathbf{IO}_t = (\mathrm{IO}_t(1), \mathrm{IO}_t(2), \mathrm{IO}_t(3))$; $\mathrm{SS}_t(i)$ is the amount of goods shipped from inventory point $i+1$ but not yet received at time period t when ordering; $\mathbf{SS}_t$ represents the vector of goods shipped but not yet received at each inventory point at time period t when ordering, i.e., $\mathbf{SS}_t = (\mathrm{SS}_t(1), \mathrm{SS}_t(2), \mathrm{SS}_t(3))$.

Fundamentally, in RL, the state summarizes all decision-related variables that have occurred before the current time step. Therefore, a well-designed state should include as much relevant historical information as possible to allow the agent to make better decisions. Thus, we define the state S_t at time t as $\mathrm{S}_t = ((\mathbf{IL}_{t-u+1}, \mathbf{IO}_{t-u+1}, \mathbf{SS}_{t-u+1}, \mathrm{D}_{t-u+1}), (\mathbf{IL}_{t-u+2}, \mathbf{IO}_{t-u+2}, \mathbf{SS}_{t-u+2}, \mathrm{D}_{t-u+2}), \dots, (\mathbf{IL}_t, \mathbf{IO}_t, \mathbf{SS}_t, \mathrm{D}_t))$, where the parameter u represents the number of steps traced back from time t (including time t).

Action

Since the problem is an ordering problem, actions can generally be set in two ways in most cases in current literature. The first method is used by Bertsekas et al. (1997)[4], where they directly use the order quantity at the inventory point as the action. This setup is simple, direct, and interpretable. However, without setting upper and lower bounds for the order quantity, the action space theoretically forms a half-space, easily leading to the curse of dimensionality, especially when data is insufficient. Kimbrough et al. (2002) [5], Tongeren et al. (2007) [7], and Chaharsooghi et al. (2008) [8] improved upon the first method by using the change in the order quantity at an inventory point with respect to the received order volume, or the vector it forms, as the action. This setup constrains the state space from an unbounded set in the previous method to a bounded set. Although this requires manually setting a boundary for the bounded set, it significantly narrows the scope of the action space, facilitating the training of RL algorithms and the search for optimal policies, especially in cases of limited data availability. However, both methods mentioned above share a notable drawback: they do not incorporate useful expert knowledge or prior information when selecting actions or making decisions, diverging from rational human decision-making processes and resulting in weak interpretability of the decision models.

Therefore, we redesign the action $\boldsymbol{a}_t$ at time t as $\boldsymbol{a}_t = (a_t(1), a_t(2), a_t(3))^{\mathrm{T}}$, where $a_t(i)\in[0,1]$. We define the column vector of order quantities at each inventory point at time t as $\mathbf{O}_t = \mathbf{X}_t\boldsymbol{a}_t$, where the order information matrix $\mathbf{X}_t = (\overline{\mathbf{O}}_t, \overline{\overline{\mathbf{O}}}_t, \widetilde{\mathbf{O}}_t)$, $\overline{\mathbf{O}}_t$ is the column vector of average order quantities at each inventory point for the previous t periods, $\overline{\overline{\mathbf{O}}}_t$ is the column vector of average order quantities at each inventory point for periods $t-2, t-1$ and t, and $\widetilde{\mathbf{O}}_t$t is the column vector of predicted order quantities at each inventory point for period t. The exponential smoothing model is used to predict order quantities in this part. It is evident that action $\boldsymbol{a}_t$ is common for all inventory points, and the product of the row vector of the i-th row of the order information matrix $\mathbf{X}_t$ and action $\boldsymbol{a}_t$ is the order quantity from inventory point i to inventory point $i+1$ at time t. Furthermore, the dimension of action $\boldsymbol{a}_t$ is independent of the number of inventory points in the inventory system, theoretically facilitating extension to any size inventory system.

Reward

Inventory systems aiming to minimize total costs, rewards are typically set as the negative of the total costs for single or partial inventory points or the entire inventory system within a single time period, as demonstrated by Bertsekas et al. (1997)[4], Giannoccaro et al. (2002)[6], and van Tongeren et al. (2007)[7]. Thus, we use the negative of the total costs for the entire inventory system within each time period as the reward.

Shaped-nStep-Double-DQN Algorithm

In the following, we present the framework of the algorithm below.

As can be seen, we employ (1) Q-learning and the idea of (2) sampling to calculate expected values in designing the objective function; moreover, we use (3) the 2-step TD method to achieve a better balance between update bias and variance, while the length of experience data does not increase significantly compared to before, thus avoiding the pressure on experience data storage. Additionally, we utilize (4) a fixed-target double network structure to reduce the impact of updates to the latest parameters on target values, thereby minimizing training oscillations and divergence. Finally, we (5) employ the reward shaping technique to stabilize training.

Table 1: Shaped-nStep-Double-DQN Algorithm

Algorithm：Shaped-nStep-Double-DQN Algorithm

Input：
Experience replay buffer capacity N；number of training episodes M；exploration rate sequence $\{\boldsymbol{\varepsilon_k}\}$，where $\boldsymbol{\varepsilon_k} \in [\mathbf{0}, \mathbf{1}]$，$\boldsymbol{k} = \mathbf{1}, \mathbf{2}, \ldots$；learning rate sequence $\{\boldsymbol{\alpha_k}\}$，where $\boldsymbol{\alpha_k} \in [\mathbf{0}, \mathbf{1}]$，$\boldsymbol{k} = \mathbf{1}, \mathbf{2}, \ldots$；discount factor $\boldsymbol{\gamma}$；number of experiences randomly sampled from mini-batch $\boldsymbol{n}$；target network parameter update frequency C；linear transformation parameter for action value $\boldsymbol{a}$；another linear transformation parameter for action value $\boldsymbol{b}$
Initialization: Experience replay buffer $\mathcal{D} = \{\}$；prediction Q network parameters $\boldsymbol{w}$；target Q network parameters $\boldsymbol{w'} = \boldsymbol{w}$
for episode =1 to M do:
  Obtain initial state s_0
  for $t = 0$: T -1 do:
  Select action a_t based on the prediction Q network (ε-greedy method)
  Execute action a_t；observe next state s_{t+1}；receive reward r_{t+1}
  Select actiona_{t+1} based on the prediction Q network (ε-greedy method)
  Execute action a_{t+1}；observe next state s_{t+2}；receive reward r_{t+2}
  Store experience data $(s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1}, s_{t+2})$ as a queue in experience replay buffer $\mathcal{D}$
  if $|\mathcal{D}| \geq m$ do:
    Randomly sample n experience data $\{(s_{(i)}, a_{(i)}, r_{(i)}, s_{(i)}', a_{(i)}', r_{(i)}', s_{(i)}'')\}_{i=1,2,\ldots,n}$
    if $|\mathcal{D}| <$ N do:

$$y_{(i)} = r_{(i)} + \gamma r_{(i)}' + \gamma^2 \mathrm{Q}(s_{(i)}'', \underset{\mathrm{a}}{\mathrm{argmax}}$$

$$(\mathrm{Q}(s_{(i)}'', a; \boldsymbol{w}); \boldsymbol{w'})$$

if $|\mathcal{D}| = \mathrm{N}$ do:
Calculate the average reward value $\bar{\mathrm{r}}$ for all experience data (2N in total) in experience replay buffer $\mathcal{D}$
Update all action values Q as $\mathrm{Q} + \frac{b}{1-\gamma}\mathrm{Q}$

$$y_{(i)} = r_{(i)} + \gamma r_{(i)}{}' - (1+\gamma)\bar{r} + \gamma^2 \mathrm{Q}(s_{(i)}{}'', \underset{a}{\operatorname{argmax}}\, \mathrm{Q}(s_{(i)}{}'', a; \mathbf{w}); \mathbf{w}')$$

else:

$$y_{(i)} = r_{(i)} + \gamma r_{(i)}{}' - (1+\gamma)\bar{r} + \gamma^2 \mathrm{Q}(s_{(i)}{}'', \underset{a}{\operatorname{argmax}}\, \mathrm{Q}(s_{(i)}{}'', a; \mathbf{w}); \mathbf{w}')$$

Minimize the loss function $\mathrm{L} = \frac{1}{n}\sum_{i=1}^{n}(y_{(i)} - \mathrm{Q}(s_{(i)}, a_{(i)}; \boldsymbol{w}))$
Update $\mathbf{w}' = \mathbf{w}$ every C iterations
$t = t + 1$
end for
end for

In the reward shaping part, we provide a rigorous mathematical explanation through Lemma below to proof the invariance of the optimal policy before and after reward transformation.

Lemma (Optimal Policy Invariance). Consider a Markov decision process with $v^* \in \mathbb{R}^{|\mathcal{S}|}$ as the optimal state value satisfying $v^* = \max_\pi(r_\pi + \gamma \mathrm{P}_\pi v^*)$. If every reward r is changed by an affine transformation to $ar + b$, where $a, b \in \mathbb{R}$ and $a > 0$, then the corresponding optimal state value v' is also an affine transformation of v^* : $v' = av^* + \frac{b}{1-\gamma}\mathbf{1}$, where $\gamma \in (0,1)$ is the discount rate and $1 = [1, \ldots, 1]^{\mathrm{T}}$. Consequently, the optimal policies are invariant to the affine transformation of the reward signals.

Proof: For any policy π, define $r_\pi = [\ldots, r_\pi(s), \ldots]^{\mathrm{T}}$ where $r_\pi(s) = \sum_a \pi(a \mid s)\sum_r p(r \mid s, a)r$, $s \in \mathcal{S}$. If $r \to ar + b$, then $r_\pi(s) \to ar_\pi(s) + b$ and hence $r_\pi \to ar_\pi + b1$, where $1 = [1, \ldots, 1]^{\mathrm{T}}$. In this case, the BOE becomes $v' = \max_\pi(ar_\pi + b1 + \gamma \mathrm{P}_\pi v')$ (*). We next solve the new BOE in (*). To do that, we verify that $v' = av^* + k1$ with $k = b/(1-\gamma)$ is the solution of (*). In particular, substituting $v' = av^* + k1$ into (*) gives $av^* + k1 = \max_\pi[ar_\pi + b1 + \gamma \mathrm{P}_\pi(av^* + k1)] = \max_\pi(ar_\pi + b1 + a\gamma \mathrm{P}_\pi v^* + k\gamma\mathbf{1})$, where the last equation is due to $P_\pi 1 = 1$. The equation can be rewritten as. $av^* = \max_\pi(ar_\pi + a\gamma \mathrm{P}_\pi v^*) + b1 + k\gamma 1 - k1$, which is equivalent to $b1 + k\gamma 1 - k1 = 0$. Since $k = b/(1-\gamma)$, the above equation is valid and hence $v' = av^* + k1$ is the solution to (*). Since (*) is the BOE, v' is also the unique solution. Finally, since v' is an affine transformation of v^*, the relative relationship among the action values remain the same. Hence, v' would lead to the same optimal policies as v^*.

Experiments

The experiments in this paper are divided into two parts:

1. Testing the performance of the Shaped-nStep-Double-DQN algorithm under various MDP settings (experiment 1).

2. Applying the Shaped-nStep-Double-DQN algorithm to a three-echelon linear inventory system (experiment 2-3).

The hyperparameters for the Shaped-nStep-Double-DQN algorithm used in this study are as follows:

Table 2: Parameters of MDP and hyperparameter Settings of Shaped-nStep-Double-DQN algorithm

Hyperparameters	Values
Number of steps to backtrack in the state	3
Experience replay buffer capacity N	20000
Number of training episodes M	2000
Number of time steps per episode T	200
Target network parameter update frequency C	20
Batch size for random sampling of experiences n	32
Discount factor γ	0.99

Initial exploration ratein ε-greedy policy ε_0	0.9
Exploration rate decay factor in ε-greedy policy	0.99
Initial learning rate α_0	0.1
Learning rate decay factor	0.99
Fully connected neural network (Q-network) structure	30 ×80×20× 3
Learning rate in RMSProp algorithm	0.01
Decay factor in RMSProp algorithm	0.9
Stabilizing factor in RMSProp algorithm	10^{-6}

Experiment 1: Performance of Shaped-nStep-Double-DQN Algorithm under Different MDP Settings

In the experiment, we fixed the reward setting and examined the performance of the MDP with different state and action settings. There are two options for the states: (1) a vector composed of the inventory level, in-transit inventory, and received orders at each inventory point, and (2) a vector composed of the inventory level at each inventory point and the shipment quantity in the past two time periods. There are also two options for actions: (1) a vector of order quantities at each inventory point, and (2) a vector of order quantity changes with respect to the received orders at each inventory point. For simplification, we denote these as state setting 1 and state setting 2, and action setting 1 and action setting 2, respectively. The action and state settings in the MDP model of this article are denoted as state setting 3 and action setting 3. We consider a specific inventory system where demand is constant, and the demand at each time period is always 5. The lead times for all inventory points are 0 (i.e., no lead time). The operational costs at each inventory point consist only of holding costs and stockout costs, both with a unit cost of 1. The external supplier has an ample supply. In this inventory system, the 1-1 ordering policy is optimal, meaning that downstream orders should match upstream orders. Thus, the optimal ordering policy is to order 5 units of goods at each inventory point. Since the lead times are 0, the inventory is received in the same time period as the order, so no holding costs or stockout costs are incurred, and the optimal ordering cost is 0. We simulated the inventory system for 20 time periods, repeating the simulation 50 times.

Table 3 shows the performance of the ordering policy learned by the Shaped-nStep-Double-DQN algorithm under different MDP settings. The values in the table are the averages of 50 experiments. As shown, the combination of state setting 1 and action setting 1, as the simplest MDP setting, results in the worst performanc, with an average total cost of 29. The MDP setting with state setting 3 and action setting 3 exhibits the best performance in minimizing inventory costs, with an average total cost of 2. This is about 0.2 times the cost of the second-best MDP setting (state setting 2 and action setting 3) and about 0.067 times the cost of the worst MDP setting (state setting 1 and action setting 1). Compared to the ordering policies under other settings, the policy learned by the algorithm with state setting 3 and action setting 3 is almost as good as the optimal 1-1 ordering policy. This further demonstrates the importance of selecting a suitable MDP model setting, improving state representation capabilities, and enhancing the relevance between actions and decisions in the learning of ordering policies.

Table 3: Performance of Shaped-nStep-Double-DQN Ordering Policies under Different MDP Settings

	State Setting 1	State Setting 2	State Setting 3
Action Setting 1	-29	-25	-23
Action Setting 2	-24	-11	-12
Action Setting 3	-16	-10	**-2**

Experiment 2: Performance between Shaped-nStep-Double-DQN Ordering Policy and 1-1 Ordering Policy

In the second part of the experiment, we applied the Shaped-nStep-Double-DQN algorithm to the linear inventory system based on the MDP model settings determined in the previous section. We set the

demand to be constant and equal to 5 for each time period. The lead time is 0 for each inventory location, and the holding cost and stock-out cost per unit are both set to 1. The external supplier is assumed to be sufficient. Figure 2 shows the training process of the learned ordering policy, with the orange horizontal line indicating the fixed reward for the optimal 1-1 ordering policy (0). The light blue curve represents the rewards obtained by the learned ordering policy in each time period. It can be seen that as the training progresses, the performance of the learned ordering policy gradually improves, and the rewards stabilize between -5 and 0, with a mean value of about -2, which is only 2 away from the optimal inventory cost of 0. This demonstrates that the Shaped-nStep-Double-DQN algorithm is able to learn an excellent ordering policy that approximates the optimal policy in a deterministic linear inventory system. The results also suggest that the Shaped-nStep-Double-DQN algorithm performs well in learning the ordering policy in deterministic linear inventory systems.

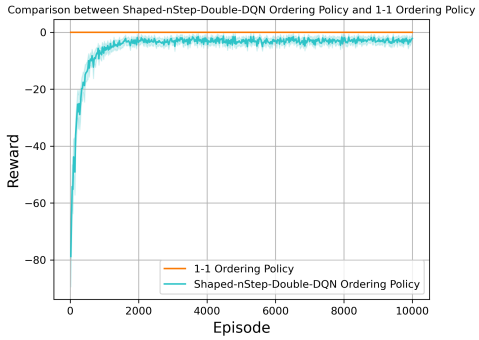

Figure 2: Shaped-nStep-Double-DQN Ordering Strategy and 1-1 Ordering Strategy Performance Comparison in Deterministic Linear Inventory Systems

Experiment 3: Performance between Shaped-nStep-Double-DQN Ordering Policy and Base Stock Ordering Policy

Next, we extend the deterministic linear inventory system to a stochastic linear inventory system. We modify the lead time of each inventory point in the deterministic linear inventory system from a constant zero to a normally distributed random variable with a mean of 0 and a variance of 1. We also change the demand at each time period, which was previously always equal to 5, to follow a Poisson distribution with a mean of 5.

Figure 3 depicts the training process of the Shaped-nStep-Double-DQN ordering strategy, where the red curve represents the rewards per time period for the echelon base-stock strategy, and the green curve represents the rewards per time period for the ordering strategy learned using the Shaped-nStep-Double-DQN algorithm. As observed, the performance of the ordering strategy learned using Shaped-nStep-Double-DQN improves during training. The rewards per time period gradually increase as the number of episodes increases, eventually stabilizing between -45 and -55, with an average around -50, outperforming the echelon base-stock strategy. This demonstrates that in this stochastic linear inventory system, the ordering strategy learned using Shaped-nStep-Double-DQN is more effective in reducing inventory costs and has better performance compared to the base-stock strategy.

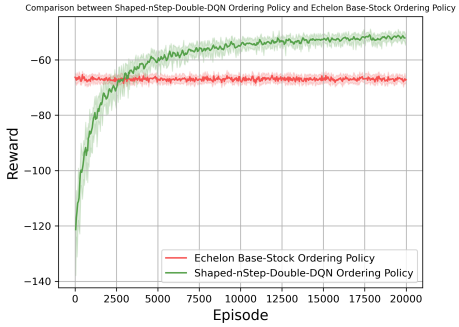

Figure 3: Comparison between the Shaped-nStep-Double-DQN Ordering Strategy and the Base-Stock Ordering Strategy

Conclusion

First of all, existing MDP models in the literature have limitations for linear inventory systems, prompting us to propose a new MDP model, which is more suitable for linear inventory system.

To determine the optimal policy for this MDP model, we introduce a new RL algorithm called Shaped-nStep-Double-DQN.In the experimental section, we design experiments and verify that our new MDP model setting outperforms the existing MDP model settings. We then test the performance of the Shaped-nStep-Double-DQN algorithm in the ordering problem of linear inventory systems.

The experimental results show that in deterministic linear inventory systems, the ordering strategy learned using the Shaped-nStep-Double-DQN algorithm is nearly consistent with the optimal ordering strategy in reducing inventory costs and is a good approximation. In stochastic linear inventory systems with Poisson-distributed consumer demand, the Shaped-nStep-Double-DQN algorithm's ordering strategy outperforms the base-stock ordering strategy in reducing inventory costs, exhibiting superior inventory performance.

References

[1] Clark A J, Scarf H. Optimal policies for a multi-echelon inventory problem[J]. Management science, 1960, 6(4): 475-490.

[2] Chen F, Zheng Y S. Lower bounds for multi-echelon stochastic inventory systems[J]. Management Science, 1994, 40(11): 1426-1443.

[3] Muharremoglu A, Tsitsiklis J N. Dynamic leadtime management in supply chains[J]. Preprint, Massachusetts Institute of Technology, Cambridge, 2003.

[4] Van Roy B, Bertsekas D P, Lee Y, et al. A neuro-dynamic programming approach to retailer inventory management[C]//Proceedings of the 36th IEEE Conference on Decision and Control. IEEE, 1997, 4: 4052-4057.

[5] Kimbrough S O, Wu D J, Zhong F. Computers play the beer game: can artificial agents manage supply chains?[J]. Decision support systems, 2002, 33(3): 323-333.

[6] Giannoccaro I, Pontrandolfo P. Inventory management in supply chains: a reinforcement learning approach[J]. International Journal of Production Economics, 2002, 78(2): 153-161.

[7] van Tongeren T, Kaymak U, Naso D, et al. Q-learning in a competitive supply chain[C]//2007 IEEE International Conference on Systems, Man and Cybernetics. IEEE, 2007: 1211-1216.

[8] Chaharsooghi S K, Heydari J, Zegordi S H. A reinforcement learning model for supply chain ordering management: An application to the beer game[J]. Decision Support Systems, 2008, 45(4): 949-959.

[9] Valluri A, North M J, Macal C M. Reinforcement learning in supply chains[J]. International journal of neural systems, 2009, 19(05): 331-344.

[10] Mortazavi A, Khamseh A A, Azimi P. Designing of an intelligent self-adaptive model for supply chain ordering management system[J]. Engineering Applications of Artificial Intelligence, 2015, 37: 207-220.

[11] Kara A, Dogan I. Reinforcement learning approaches for specifying ordering policies of perishable inventory systems[J]. Expert Systems with Applications, 2018, 91: 150-158.

[12] Boute R N, Gijsbrechts J, Van Jaarsveld W, et al. Deep reinforcement learning for inventory control: A roadmap[J]. European Journal of Operational Research, 2022, 298(2): 401-412.

[13] Goedhart J, Haijema R, Akkerman R. Modelling the influence of returns for an omni-channel retailer[J]. European Journal of Operational Research, 2023, 306(3): 1248-1263.